System-Level Hardware/Software Trade-offs

Samuel P. Harbison
Texas Instruments
300 Oxford Drive
Monroeville, PA 15146
+1 (412) 380-1547
s-harbison@ti.com

1. ABSTRACT

Operating systems and development tools can impose overly general requirements that prevent an embedded system from achieving its hardware performance entitlement. It is time for embedded processor designers to become more involved with system software and tools.

1.1 Keywords

Digital signal processors, instruction set architecture, compiler, real-time operating system, software configuration.

2. INTRODUCTION

Embedded processors are typically evaluated on instruction execution speed, I/O bandwidth, and power consumption. The actual performance of an embedded system depends on many other factors, including how well the instruction set architecture (ISA) can be exploited by software.

Efficiency is particularly important in specialized real-time computers such as digital signal processors, where all available performance is needed for new applications such as AC-3 audio decoding or DSL data communications. Efficiency directly impacts total system cost when, for example, vendors compete on how many V.90 modems can be supported by a single DSP. This level of efficiency is less important on general-purpose processors such as PCs, where programmers are less exposed to the hardware.

There are several obstacles to realizing the available performance. First, as system software is increasingly supplied by independent vendors, hardware designers and software designers rarely work together, and their technologies develop independently rather than synergistically. Second, there are many embedded processors and few common software design frameworks, so system software vendors must support many designs and processor families with general solutions. Finally, product designers worry more about large software building blocks; there is little mind-share and few royalty dollars left for system software.

This paper looks at some ways in which hardware designers can improve system performance using system software and tools. This is not an exhaustive list, but may stimulate ideas.

3. INSTRUCTION SET ISSUES

With software being written mostly in high-level programming languages, instruction set design must be coordinated with optimizing compiler technology. New ISA technologies may in fact be gated by compiler research. For example, the market acceptance of TI's VLIW architecture in its TMS320C6000 series of DSPs is due in part to the success of software pipelining optimizations in TI's compilers.

Compilers can also enable ISA features that would have been unacceptable when software was written entirely in assembly language. For example, computers with non-interlocked pipelines are extremely difficult to program in assembly language, because the programmer must keep track of the execution progress of many instructions at the same time. TI addressed this issue by providing a "linear assembly language" that presents the same instructions as the C6000, but hides pipeline parallelism from the programmer. The assembler optimizes linear code for the actual ISA.

Specialized instructions on DSPs also present compiler issues. There are several cases:

- Instructions used only in a few algorithms should be isolated in specialized assembly code libraries. They don't affect software tools.
- Optimized instructions, such as multiply-accumulate, should be generated by an enhanced compiler from common programming idioms such as array indexing.
- ISA features such as circular addressing or fixed-point arithmetic may be best accommodated with language extensions. Alternatively, the compiler can provide direct access to the instructions through assembly inserts or intrinsic functions.

4. OPERATING SYSTEM ISSUES

With so many embedded processor families and so many ways of designing real-time software, a real-time operating system (RTOS) must mediate between two sets of abstractions. On the hardware side, it must match the architectures of embedded processors from several different manufacturers. On the software side, it must support real-time applications with wildly differing requirements and software architectures. The result is often a generalized RTOS that is a poor fit to either side.

4.1 Software Configuration

Most general-purpose operating systems are designed to support dynamic work loads, but embedded processors traditionally run a fixed application. Placing one-time RTOS initialization code on the processor uses memory and cycles without purpose. What is needed is a “resource editor”—a tool that supports the static configuration of software objects when the application is built. These tools are commonly available in PC software development to support fixed graphical objects.

For DSPs, TI’s DSP/BIOS(tm) software is designed to be statically configured as part of the application development process. A configuration editor is used to define and configure the tasks, device drivers, and communication channels needed. Pre-allocated data structures are linked with the target application, and a corresponding collection of debug and analysis information remains on the host to help in debugging or monitoring the application.

4.2 Addressing and Protection

Despite what was just said about fixed applications, designers are planning more ambitious embedded computer products. Customers will be able to customize their video centers and cellular phones by adding new software applications. The success of this approach depends in part on making it easy for independent software vendors to write such added-value applications, while protecting the products’ basic functionality from bugs in those applications.

The traditional hardware solution is to provide protected virtual memory on the embedded processor, simplifying programming and isolating applications from each other. Microsoft WinCE(tm) and other general-purpose operating systems being adapted for embedded use depend on such hardware support.

The performance cost of additional logic for hardware-assisted virtual memory is unacceptable in digital signal processors. TI’s approach has been to combine a DSP with a general-purpose processor on a single chip. The general processor provides a protected environment for software applications and the larger OS. The DSP handles the hard real-time tasks with higher performance and lower total power consumption than the general processor could achieve alone. Supporting such a system requires a careful hardware system architecture, plus investments in tooling and system software for I/O and communication. The hardware vendor is in the best position to provide this support.

4.3 Devices and Drivers

Peripheral devices are another area in which hardware opportunities and application requirements are negotiated by an independent RTOS vendor. Hardware designers often include advanced features in their peripheral devices to get higher throughput. These features will be useless if the operating system’s device drivers will not take advantage of them, either because the driver model is too simple or because the device design is incompatible with higher-level operating system abstractions (e.g., multiprocessing).

Designing the drivers and devices together, including the application interfaces to the drivers, yields the best performance. Properly designed drivers can also demonstrate how a device *ought* to be used—something normally relegated to application notes.

5. REAL-TIME ANALYSIS AND DEBUG

Hardware designers can also help application developers get their products to market more quickly. Integration, debugging, and analysis of real-time systems can be very difficult, especially in the final product stages.

The simplest hardware tool needed to support performance analysis is an instruction or cycle counter. Unfortunately, many embedded processors supply only a general-purpose, periodic timer, which is often too narrow (e.g., 16 bits) and too difficult to share with other applications. (Stopping and starting the timer can degrade applications with clock skew.) Hardware designers could support measurements by providing a simple, nonstop, read-only cycle counter at least 32 bits wide. Using this device, programmers can collect good performance data with minimal overhead.

More extensive monitoring of real-time system depends on the ability of a development host to send and receive data from the target system unobtrusively. Software techniques using low-priority tasks to transfer data over serial ports are not useful in high-performance DSP systems; they are too intrusive at the data rates needed to monitor real-time signals. TI’s approach is to provide RTDX(tm), hardware-supported data transfers using the JTAG debug port.

Finally, hardware-assisted, stop-mode debugging of real-time systems is not useful when the target cannot fail to respond to interrupts controlling external actions. Real-time debug facilities are needed that can halt the DSP while still allowing designated interrupts to be processed. TI is integrating this capability into its DSPs and is supporting it with extensions to the debugging and analysis tools.

6. ACKNOWLEDGEMENTS

Many of the techniques in this paper have been developed by my colleagues in TI’s Software Development Systems. I especially want to thank Bob Frankel, Ed Kuzemchak, Bob McGowan, Dave Russo, Lee Szewerenko, and Reid Tatge.

7. REFERENCES

[1] DSP/BIOS General Overview. URL http://www.ti.com/sc/docs/dsps/tools/dspbios/index.htm.

[2] TMS320C600 product information. URL http://www.ti.com/sc/docs/dsps/products/c6000/index.htm.

[3] RTDX. URL http://www.ti.com/sc/docs/dsps/tools/c5000/c54x/rtdx.htm.

[4] “Emulation Fundamentals for TI’s DSP Solutions.” URL http://www.ti.com/sc/docs/psheets/abstract/apps/spra439.htm.